BECA: A Blockchain-Based Edge Computing Architecture for Internet of Things Systems

IoT

MDPI

Article

BECA: A Blockchain-Based Edge Computing Architecture for Internet of Things Systems

Oluwashina Joseph Ajayi [1,*], Joseph Rafferty [1], Jose Santos [2], Matias Garcia-Constantino [2] and Zhan Cui [3]

[1] British Telecom Ireland Innovation Centre (BTIIC), School of Computing, Ulster University, Jordanstown Campus, Shore Road, Newtownabbey BT37 0QB, UK; j.rafferty@ulster.ac.uk
[2] Pervasive Computing Research Group, School of Computing, Ulster University, Jordanstown Campus, Shore Road, Newtownabbey BT37 0QB, UK; ja.santos@ulster.ac.uk (J.S.); m.garcia-constantino@ulster.ac.uk (M.G.-C.)
[3] Converged Network Security Research, BT Applied Research, Adastral Park, Martlesham, Ipswich IP5 3RE, UK; zhan.cui@bt.com
* Correspondence: ajayi-o2@ulster.ac.uk

Abstract: The scale of Internet of Things (IoT) systems has expanded in recent times and, in tandem with this, IoT solutions have developed symbiotic relationships with technologies, such as edge Computing. IoT has leveraged edge computing capabilities to improve the capabilities of IoT solutions, such as facilitating quick data retrieval, low latency response, and advanced computation, among others. However, in contrast with the benefits offered by edge computing capabilities, there are several detractors, such as centralized data storage, data ownership, privacy, data auditability, and security, which concern the IoT community. This study leveraged blockchain's inherent capabilities, including distributed storage system, non-repudiation, privacy, security, and immutability, to provide a novel, advanced edge computing architecture for IoT systems. Specifically, this blockchain-based edge computing architecture addressed centralized data storage, data auditability, privacy, data ownership, and security. Following implementation, the performance of this solution was evaluated to quantify performance in terms of response time and resource utilization. The results show the viability of the proposed and implemented architecture, characterized by improved privacy, device data ownership, security, and data auditability while implementing decentralized storage.

Keywords: IoT; edge computing; auditability; Blockchain; non-repudiation; privacy; security

Citation: Ajayi, O.J.; Rafferty, J.; Santos, J.; Garcia-Constantino, M.; Cui, Z. BECA: A Blockchain-Based Edge Computing Architecture for Internet of Things Systems. *IoT* **2021**, 2, x. https://doi.org/10.3390/xxxxx

Academic Editors: Benoît Parrein and Bastien Confais

Received: 6 September 2021
Accepted: 11 October 2021
Published: 13 October 2021

1. Introduction

Over the years, the Internet of Things (IoT) has evolved across different facets of our lives. This ranges from its usage in homes, healthcare, and supply chain, as well as for industrial purposes. It has been projected that the adoption of IoT devices will increase, reaching approximately 75 million devices connected by 2025 [1]. This increase in the number of devices may be attributed to diverse deployments of IoT solutions across many application domains, including smart homes [2], healthcare [3], smart grid [4], smart cities [5], agriculture [6], and supply chain management [7].

Irrespective of the use case, IoT systems have a great potential to enhance both the functionalities and capabilities of the service they provide or support [8]. However, these use cases and emerging new ones will result in new dimensions of challenges. A major challenge is centralized data storage. As the number of IoT devices increases, the amount of data aggregated in a single location, as well as the current storage mode in the present-day implementation of IoT systems, also increase. This central data aggregation will also result in data management and data ownership issues. Ultimately, data aggregation in a central location could become attractive to attack, exploiting the IoT system to cause major damage. This becomes a security and privacy concern within the IoT ecosystem [9]. IoT systems need

to effectively and safely interact and integrate with their environment; therefore, a system that can leverage the existing capabilities of edge computing platforms and resolve some of these issues is required to enable the scalable connected future [10].

Such IoT solutions generate a vast quantity of data subject to instant processing and analysis to support decision-making. To optimize responsiveness and reduce bandwidth requirements, such processing should happen at the edge of the network. Edge computing can process data at the edge of the network and make autonomous decisions [10]. Consequently, the strength and weaknesses of edge computing concerning IoT must be explored.

1.1. Overview of Edge Computing

IoT systems can generate vast quantities of data from diverse types of IoT devices. The data can be processed at the edge of the network to improve responsiveness and scalability before being transferred to the cloud. Edge computing can transfer data from the edge of a network to the cloud's network and other resources, such as storage capabilities and intelligent services. These capabilities can provide the critical services needed by real-time solutions while meeting IoT systems requirements, such as high bandwidth and low latency on the edge network [11].

Edge computing has several variations designed for distinct purposes, such as mobile computing and ad-hoc networks. These variations include multi-access edge computing (MEC) and vehicular ad-hoc networks (VANETs). Multi-access edge computing (MEC), formerly known as mobile edge computing, enables information technology (IT) services environment with cloud capabilities at the edge of the cellular network [10]. Furthermore, vehicular ad-hoc networks (VANETs) enable groups of cars to maintain communication across the edge of the network [12].

Despite the numerous advantages of edge computing, there are open challenges, such as centralized data storage, which can lead to a single point of failure and a point of attack that can be exploited by adversaries [13], as well as IoT user data ownership.There are also concerns about who the data generated by IoT devices belongs to, i.e., the user or the edge computing platform providers. Data or information security and privacy issues [10], and even data auditability, which can support non-repudiation, among others [9] are also of great concern.

Several studies have investigated the usage and effectiveness of edge computing. Jiang et al. [14] explored the energy efficiency of edge computing as it relates to edge devices and servers. Lan [15] leveraged edge computing capabilities for real-time monitoring of a complex IoT-based event that involves data communication and processing mechanism. In an IoT study, edge computing was also used based on a unified platform for heterogeneity sensing between devices by Lan et al. [16]. These studies did not attempt to address any edge computing limitations or deficiencies but leverage its features.

Edge computing has some features that make it ideal for different real-time applications, such as smart home and smart monitoring. These features include location awareness, proximity to users, mobility support, ultra-low latency, dense geographical distribution, and interoperability [17]. With edge computing gaining attention, in the review by Liu et al. [18], some edge computing systems were identified, including CORD, Akraino Edge Stack, EdgeX Foundry, Apache Edgent, and Azure IoT Edge. These applications are now being widely used in the IoT domain, for testing, and in production.

Amongst all the identified features of edge computing, its limitations, for example the centralized data storage, cannot be underestimated and, therefore, need to be examined to provide adequate solutions and appropriate steps to deal with them. These steps start by examining the current edge computing architecture within the IoT domain to provide a robust architecture that will address these limitations.

1.2. Overview of Distributed Ledger Technology

DLT can be described as a decentralized system with multiple nodes, which are usually peer-to-peer networks. These nodes can save and keep immutable records on a database called Ledger [19]. In 2009, the first known DLT, called Bitcoin [20], became popular for its digital currency value, and it forms the basis for what we now know today as blockchain technology. Different types of Blockchain can be deployed based on two main strategies: permissionless and permissioned.

Permissionless ledgers can allow any entity to create and validate blocks in the ledger. Entities can also update transactions at will without requiring any permission within the network. This provides an open, transparent, and accessible set of transactions between all entities, and as a result, privacy within the network becomes questionable.

On the other hand, a permissioned ledger preserves privacy within a network. Members within this network are uniquely identified, resulting in all entities being properly authorized and trusted. These features make this type of blockchain relevant in some use cases, especially those requiring high security and privacy.

Permissioned blockchain has some inherent features which give it an edge over centralized systems. These include decentralization, privacy, immutability, security, scalability, accountability, data auditability, and data providence. Today, there are different types of blockchain, some of which are developed and designed for specific purposes [21–26]. With these features of blockchain, if it is successfully combined with the edge computing platform, some of the deficiencies of the current IoT representation will be adequately addressed. Edge computing will provide the required high processing power and reduced latency, while blockchain will provide the IoT systems with the features, such as the following: security, privacy, data auditability, non-repudiation of action among IoT devices, and, most especially, decentralized storage.

Hyperledger Fabric [27] (Fabric) has been identified as an enterprise blockchain network and has been used by many organizations for different use cases [28]. The uniqueness of Fabric lies in its default design tailored towards privacy and maintaining multiple copies of consistent ledgers across the peers within its network. These features make Fabric stand out and, therefore, will be used within our implementation.

1.3. Goals of the Study

The focus of this study is to develop, design, and implement a solution that addresses the identified edge computing deficiencies or limitations while leveraging on its capabilities. Below are our contributions to knowledge:

1. To provide a novel blockchain-based edge computing architecture that leverages edge computing and blockchain capabilities to offer a scalable, secure, and distributed IoT System.
2. To provide a distributed IoT System that supports data privacy, data auditability, and non-repudiation of actions between IoT devices.
3. To provide a custom blockchain network adapter that automatically creates the blockchain network and facilitates the connection to the edge computing platform through a designed and implemented middleware.

These goals are achieved by developing and designing our architecture based on edge computing and blockchain with proof of concept (PoC) implementation. Furthermore, the results from the communication between the edge and blockchain platforms were also presented, and performance testing of our implementation (PoC) was carried out.

The rest of this paper is organized as follows. Section 2 focuses on the background and related work, starting by discussing the need for decentralized storage in IoT systems, followed by a review of existing architectures. Section 3 focuses on our proposed blockchain-based edge computing architecture alongside a discussion that focuses on the detailed components of our architecture. Section 4 discusses our implementation setup and

results, while performance testing is presented in Section 5. We present our conclusion and future work in Section 6.

2. Background and Related Work

A major challenge in the current representation of IoT systems is the centralized nature of the platforms currently being used in different use cases, leading to a single point of failure [29]. Any availability issue related to such a system can lead to extreme damage and consequently downtime to the entire network and ecosystem. Other identified challenges include data privacy, security, data ownership, data auditability, and non-repudiation of actions between IoT devices. These challenges may be addressed by exploring a distributed IoT systems representation [13]. Data can be stored and processed on different entities in a distributed system, and IoT devices can securely communicate to exchange data without centralized entities [30]. Creating a distributed representation of IoT systems requires a distributed ledger with the following inherent properties: transparency, privacy, immutability, auditability, and security [31].

Interoperability plays an important role within IoT systems as it enables and supports communication between heterogeneous IoT devices. To this end, interoperability issues need to be considered while exploring the efficiency of distributed approaches in resolving the identified challenges of edge computing. At the device interaction level, edge computing platforms can provide interoperability, which is essential in IoT systems by enabling the design and implementation of different communication protocols [14]. For example, a provision of protocols such as Bluetooth Low Energy (BLE), message queuing telemetry transport (MQTT), Modbus, and simple network management protocol (SNMP) on a single Edge Computing platform. In addition, there are enterprise-grade edge computing platforms in existence today that support interoperability; an example of this is EdgeX Foundry [32]. EdgeX Foundry enables devices of different communication protocols to communicate on the edge network.

2.1. Edge Computing IoT Architecture

Currently, no global standards for IoT architecture can easily be referenced, and researchers have suggested different architectures for IoT systems [33]. Some of these architectures may align with a specific use case or application. For instance, Martin et al. [34] proposed and designed an architecture based on edge computing and edge smart gateways. Their architecture was designed to filter and aggregate data within their application and was tailored towards a structural health monitoring of IoT applications to reduce latency. In addition, their architecture provided a centralized location for data aggregation, which can disrupt the availability of the IoT solution if exploited.

In another study by Alanezi and Mishra [35], an architecture based on edge computing and an IoT framework was used to devise and execute an ambient intelligent task such as the automated discovery of new sensors and services at the network's edge. In their approach, mobile devices use Bluetooth Low Energy (BLE) to discover services offered by the edge server. These services are then shared with available sensors within the environment for planning purposes. They achieved an IoT-based architecture using edge computing, but data privacy was impossible because all the devices store data and share resources and services at the network's edge.

Cicirelli et al. [36] designed a cognitive-enabled edge-based IoT (CEIoT) architecture for smart environments, which exhibit cognitive behaviors. This architecture was designed to support cognitive computing in a distributed context, usually for edge-based computational nodes. In addition, their architecture was designed to address issues of data computation, which was achieved at the edge of the network; however, data are still centrally stored, with no privacy maintained within the represented smart environments.

Gheisari et al. [37] presented a simulation of an edge computing-based architecture designed for privacy-preserving of an IoT-based smart city. They used ontology within

the architecture of the network's edge to support the conversion of a highly dynamic mode to privacy behavior aspect entities. As a result, they could achieve privacy with their approach, but data were still centrally stored, and devices could not be held accountable for the data they shared or transmitted. These are major issues because this architecture was proposed within the smart city context, with many IoT devices.

Other studies leverage the capabilities of edge computing to design their architecture for different purposes. These include a study by Marjanovic et al. [38], which presented a reference architecture for mobile crowdsensing (MCS) deployments. They used their approach to decentralize MCS services to improve performance in terms of scalability. However, privacy issues, alongside data ownership and security, were not adequately addressed within their solution. An edge-based architecture proposed by Pace et al. [39] supports healthcare applications within Industry 4.0. The literature noted that edge computing capabilities were adopted for one use case or the other with no adequate attempt to address its challenges. Some of the challenges that were not addressed were centralized data storage, data security, and privacy [40].

2.2. Blockchain-Based Edge Computing IoT Architecture

To address some of the challenges of edge computing identified in previous sections, some researchers have explored blockchain technology. For example, Akkaoui et al. [41] designed an edge computing and blockchain system called "EdgeMediChain". Their architecture implemented using Ethereum—is a form of permissionless blockchain used to facilitate scalability and security within the healthcare ecosystem. However, their approach used a permissionless blockchain that does not support privacy and does not uniquely identify the entity within its network.

In another study by Bonnah and Shiguang [42], edge computing and blockchain were used to achieve decentralized security by eliminating a public trusted entity. In their approach, notable principles of permissioned blockchain were used within the network. They also achieved authentication of users within the solution that intends to access a service or resources. In addition, their approach addressed issues of a single point of failure by providing distributed data storage, but privacy was not adequately considered and dealt with.

Chuang et al. [43] introduced "TIDES", a trust-aware IoT data system that relies on blockchain and multi-access edge Ccmputing. Their solution, which is economic in nature, allows IoT devices to trade data and reduce trading latency. However, this study was not focused on privacy among IoT devices. Furthermore, data storage is still partially central on the multi-access edge computing platform. A study by Cui et al. [44] proposed and designed a trusted edge computing IoT platform based on blockchain. Their work emphasized solving task allocation problems within IoT environments through a heuristic algorithm. Some studies by Guo et al. [45,46] also adopted the combination of edge computing and blockchain to design a selected construct collaborative mining network (CMN) within IoT mobile devices and also created a distributed and trusted authentication system. This was achieved by applying edge computing to the blockchain node to provide name resolution, authentication service, and collaborative sharing.

Other concepts are based on the integration of blockchain and edge computing. These include concepts based on deep reinforcement learning [47], artificial intelligence [48,49], and video surveillance systems [50]. Most of these studies used permissionless blockchain, which does not provide adequate privacy. This calls for the design, development, and implementation of an architecture that uniquely leverages enterprise-grade edge computing and permissioned blockchain platforms to achieve security, privacy, data auditability, and distributed storage.

It was established here that an architecture based on edge computing and blockchain is necessary, but most importantly, the studies examined do not address the major challenges identified. Therefore, this study will explore and use Hyperledger Fabric identified in Section 1.2 to provide a solution that will remove the identified challenges.

3. Proposed Architecture

The purpose of architecture is to achieve an efficient representation of a system. In the IoT context, architecture is required to represent, organize, and present the functional structure of the IoT systems for efficient functionalities. These functionalities include supporting the hardware, software, workflow, network, protocols, services, and applications. IoT architectures can have three, four, five, or six layers [40]. This study adopts a three-layer architecture, as shown in Figure 1, which was also conceptualized to include the major components of IoT representation.

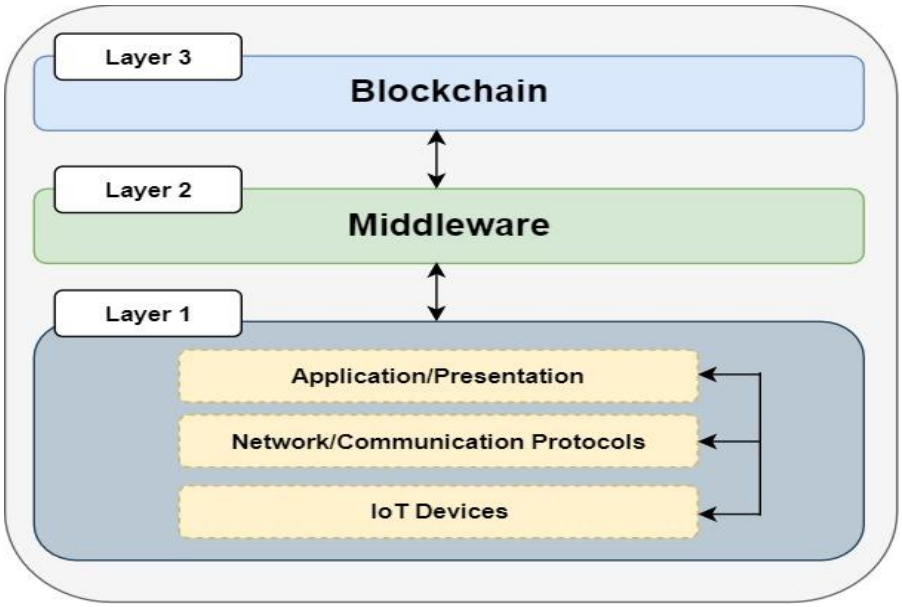


Figure 1. A three-layer annotated architecture.

The first layer (layer 1 at the bottom) replicates the full stack of an edge computing system. The IoT devices represent any device connected to the Internet and can acquire or transmit data. The network or communication protocols are essential for communication among IoT devices. This enhances and provides interoperability between heterogeneous devices or elements of the IoT System. This layer's application or presentation segment provides a user interface (UI) to enhance visualization within the IoT systems.

The middleware layer (layer 2) provides a connection between layers 1 and 3. Typically, this layer may consist of technologies that connect these two layers to facilitate data transport. With this layer in place, a connection can automatically be achieved between layers 1 and 3.

The blockchain layer (layer 3) provides a decentralized storage capability. Depending on the design approach and implementation steps taken, it can also provide security, privacy, immutability, and data auditability. To conceptualize the annotated architecture presented in Figures 1 and 2, details of the internal components in each of the three layers of the proposed architecture with technologies adopted at each layer are proposed. Generally, to actualize the proposed architecture's design, development, and implementation, enterprise-grade edge computing and blockchain were adapted for layers 1 and 3, respectively. In addition, for layer 2, custom microservices and a message bus were designed and implemented to establish and facilitate the connection between layers 1 and 3. The technologies adopted in our proposed overall architecture in Figure 2 are briefly described.

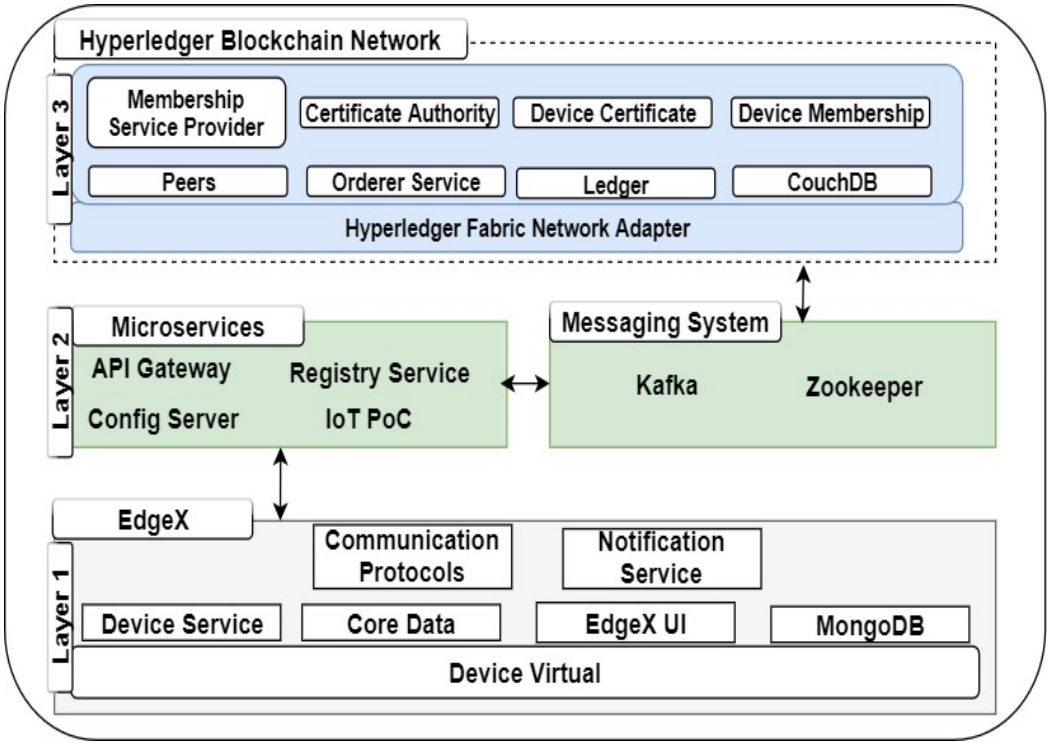


Figure 2. Proposed overall architecture.

1. Layer 1—EdgeX Foundry

In layer 1, EdgeX Foundry [32] (EdgeX), an open source IoT Edge computing platform, which supports interoperability within heterogeneous devices, was used. The EdgeX platform is modular in nature and is microservice-based, with several services within it connected by an application programming interface (API). The microservices can be altered, adjusted, or plugged together with a similar proprietary component from another vendor based on a user-defined application. This property makes EdgeX a vendor-neutral edge computing IoT system.

As shown in Figure 2, components of EdgeX were harnessed with the distributed nature and features of blockchain to remove identified challenges in current representations of IoT, such as centralized data storage, amongst others. The components of EdgeX used were device virtual, device service, data storage, notification service, core data, and communication protocols. These are sets of microservices that were selected to achieve layer 1.

2. Layer 2—Microservices, and Apache Kafka

The middleware consists of four microservices and Apache Kafka. The four microservices developed include the config server, the registry service (discovery service), the API gateway (Zuul), and the IoT PoC. Our microservices were developed using the Spring Framework based on Java and using Maven as the build tool. Spring Framework is used to develop a Spring Boot application. Each microservice exposed a port for communicating with each other. Since Spring application requires a database path to facilitate communication, we used MySQL as our Java Database Connectivity (JDBC) and runs on port 3306. Table 1 summarises the features of the four microservices.

Table 1. Microservices and their overall properties.

Microservice Name	Spring Application Name	Server Servlet.Context Path	Server Port
Config Server	Iotconfigerver	-	9003
API Gateway	Zuul	/api-gateway	9000
Registry Service	Discoveryservice	-	8010
IoT PoC Service	Iotpocservice	/api-poc	9001

Apache Kafka is the message bus used within the proposed architecture. The need for a robust, distributed messaging, fault-tolerant, and scalable system gives rise to the development of Apache Kafka [51] by Linkedln before it was moved to the Apache Software Foundation. Kafka can be used as a message bus or message center that provides data fields between different applications. It leverages its topics, group, and cluster resources to provide a publish-subscribe service within participating applications. There are typically data provider(s) and data consumer(s) in such a system.

Kafka uses Zookeeper, which provides synchronization and flexibility within a distributed system to manage its cluster node, topics, partitions, and other delegation activities. In addition, for the design and implementation of our architecture, Apache Kafka and Zookeeper serve as a message bus between Hyperledger Fabric and EdgeX through our microservices.

3. Layer 3—Hyperledger Blockchain Network

Fabric offers a range of advantages over other nlockchain systems. Some of these include maintaining multiple ledgers across its peers and its modular nature as some of its components can be replaced. For instance, the certificate authority (CA) can be replaced by any third-party X.509 compliant CA. In addition, the consensus mechanism that ensures the correctness and consistency of the internal state of the ledger is fully pluggable. Another unique feature is the smart contract called chaincode, which can automatically execute specific instructions tailored towards a particular use case or scenario.

Figure 3 depicts a sample of the Hyperledger Fabric Network used, supporting the one depicted as layer 3 in Figure 2.

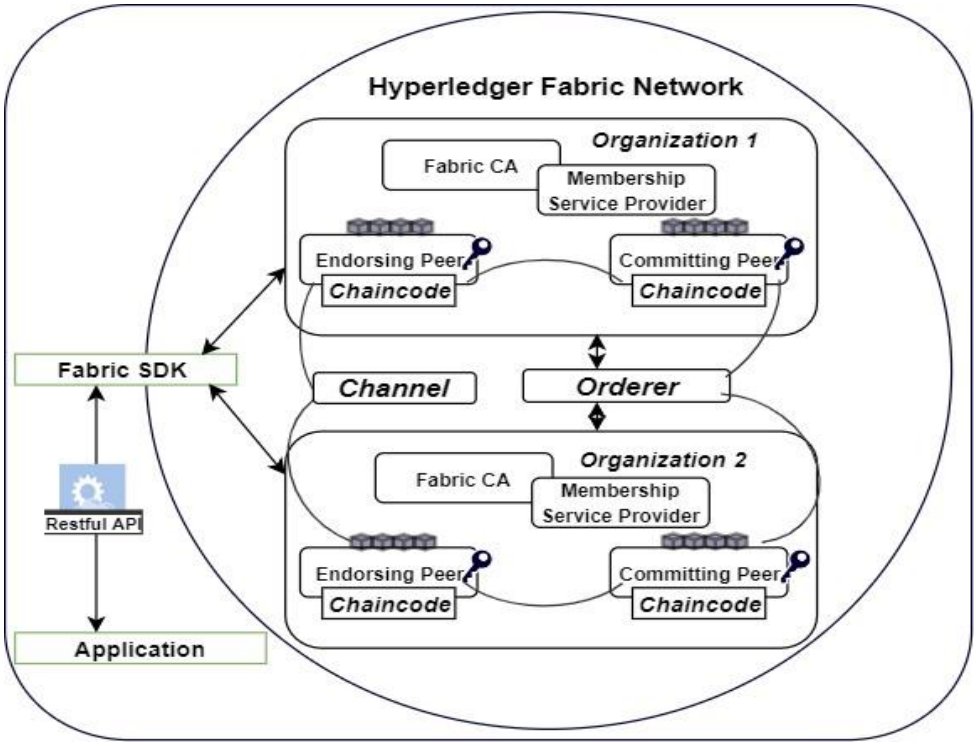


Figure 3. Hyperledger Fabric Blockchain Network.

Fabric Software Development Kit (SDK): Currently, Fabric supports NodeJS, Java, and Golang. The user application can interact with the SDK through Restful API with transaction proposals submitted through the peers of each organization within the network. Organizations within the network are units representing an entity. For example, an entity can be a company or IoT device associated with a Fabric certificate authority (Fabric CA).

Membership Service Provider (MSP): To maintain security and identity management, each organization has an MSP that uniquely identifies its users; thus, the MSP of an

organization is unique from the other. The MSP, through the Fabric CA, generates identities for the users. For example, an admin user is generated by default once a network is instantiated. However, the number of other users depends on the parameters specified while setting up the network.

Channel: This is used by Fabric to maintain privacy within the network. Two or more organizations can have a channel created between them while having another channel for interacting with each other. This ensures data privacy and integrity by making data available to the intended users within the organization.

Peers: Peers within organizations are responsible for proposing transactions detailed in the chaincode. Each organization has multiple peers (Endorsing/Anchor, Commuting, and normal). The peers communicate with each other using gossip protocols while keeping the content of the ledger consistent.

Orderer: This is responsible for creating, committing, and maintaining blocks in the ledger. The ledger which is connected to each peer is a persistent storage within the network. This persistent storage across the ledger of the peers gives Fabric the decentralized storage feature.

In a general context, Figure 3 depicts the following: through the MSP and Fabric CA, all entities within an organization are registered. Channel(s) are instantiated to maintain privacy between two or more organizations. Fabric also supports creating an endorsement policy that further maintains security and privacy. A chaincode is instantiated on the Peers. Ideally, a peer that is visible to other organizations is called the Anchor or Endorsing peer. The Fabric uses a database (CouchDB) to persist the ledger. The contents of the ledger on each peer are consistently maintained.

Overall, the functionalities of Fabric include identity management, which is achieved through its MSP and CA. Furthermore, privacy, security, and confidentiality are maintained by creating channel(s) and endorsement policy. The orderer ensures that there is efficient processing which provides scalability and consistency within Fabric. Finally, modularity is achieved by supporting pluggable CA and consensus algorithm that determines how the orderer commits blocks to the ledger through the peers. All these features match the criteria for what we intend to achieve, which justifies why we use Fabric within this study.

Similarly, some studies have leveraged the modular nature and capabilities of EdgeX to achieve their aim. For example, Kim et al. [52] leveraged EdgeX UI and analytical features and used it as a monitor platform in their work based on Deep Q-Network. Sobecki et al. [53] also leveraged the EdgeX platform for visualization in their work based on deep learning. Others [54,55] also used this platform in one way or the other. However, none of these studies proposed an architecture based on EdgeX and Fabric platforms, which is one of the major steps taken in this study.

4. Implementation and Results

This section describes the steps taken in the design, development, and implementation of the proposed architecture. First, the identified components within the architecture were presented to align with how they were used within the implementation, followed by our implementation approach. Finally, this section closes with the presentation of the results from the implementation.

4.1. Implementation Components—Specifics

The major components of our implementation are presented here to show the specific steps taken. We started from layer 1, which is the EdgeX platform, through to layer 3.

Layer 1: EdgeX

EdgeX Foundry, implemented in the Go programming language and as a set of microservices, is also containerized and runs as Docker containers. As with every Docker-related application, EdgeX has a YAML file, and an instance of this can be run as required.

We selectively enabled some major components that are required for our study. Two virtual IoT devices were enabled through the edgex-device-virtual services running on port 49990 with a dedicated API. Other enabled components are coredata on port 48080, metadata on port 48081, and command on port 48082. This selection makes our version of EdgeX lightweight and runs faster. The UI, which runs on port 4000, was used for logging and virtualization. MongoDB was also used for storage and data persistence. EdgeX assigns unique objectID to devices for standard and proper identification. This formed layer 1 of the proposed architecture (edge computing), which provides high processing power, low latency, and interoperability among IoT devices. Table 2 gives a description and details of the selected EdgeX microservices used within our implementation.

Table 2. Selected EdgeX microservices description and their respective port number.

S/N	Microservice	Description	Port
1	edgex-device-virtual	This microservice simulates different types of devices to generate events and provide readings to the coredata microservice.	49990
2	coredata	It provides centralized persistence of all the data collected by the devices.	48080
3	metadata	The metadata stores information relating to a device.	48081
4	command	This microservice enables the issuance of commands and action to devices, usually through the GET and PUT commands.	48082
5	edgex UI	EdgeX UI provides the interface to manage and monitor an instance of EdgeX.	4000
6	MongoDB	EdgeX uses MongoDB to persist both the data and the metadata of (connected) devices	27017

Layer 2: Microservices

Briefly described here are the four microservices within Layer 2 that are used in the implementation.

1. Config Server Microservice: In a distributed system such as this, Spring Cloud Config supports and provides externalization of configurations common to all the microsystems within the solution. It provides a collective place where all application's external properties can be managed. This also helps to secure the application as these common files are usually located outside the main application directory, and the path is also encrypted. All microservices will not run until these files are 'called' within each microservice. For our implementation, we used the 'Desktop/dev' path and the 'genkey' command of the 'keytool' utility tool to generate a Keystore that supports the Secure Shell layer (SSL). This provides a secure design approach for our implementation by ensuring that a proper connection is established at runtime.

2. API Gateway Microservice: In Spring applications, the API gateway selected was developed by Netflix, called Zuul. In our implementation, Zuul is used to achieve authentication by identifying various resource requirements and declining requests that do not meet these requirements. It also provides dynamic routing to other microservices hosting the requested services. Zuul was integrated with Eureka (on port 8010) and hosted on our registry service, which helps to identify microservice network locations by dynamically discovering them within the network. During bootstrap, it calls the Config Server microservice to enforce its profiles and to enable access. It also provides authentication and authorization to external users before devices are registered, and data persisted to MongoDB of EdgeX.

3. Registry Service Microservice: This microservice is based on Eureka, which is a discovery service. Each microservice registers itself on this while specifying its host, port, and node name, as well as other specific metadata. Other microservices can use this

metadata in making important decisions. Multiple instances of other microservices can be created here to provide fault tolerance to the solution.

4. IoT PoC Microservice: This deals with the major functionalities of the implementation. It was used for communication from EdgeX, through Kafka, to Hyperledger, and vice versa. It facilitates the registration of devices on our local database (MongoDB) provided by EdgeX. After the registration, the devices automatically get registered on Fabric using the APIs designed as a 'Hyperledger network adapter. Its 'application. properties' specified the JDBC database host, port, name (MySQL, in this case), and the Kafka details (group_id, admin, producer, and consumer bootstrap-server port which runs on port 9092). This microservice was also used to implement data transfer objects (DTOs) that maps to the functionalities of EdgeX, Kafka configuration, Swagger configuration (which helps to interact with the API gateway), and other resources. Finally, its bootstrap properties referenced the Config Server microservice for security enforcement.

Table 3 provides a brief description of the four microservices used within Layer 2.

Table 3. Selected EdgeX microservices description and their respective port number.

S/N	Microservice	Description
1	Config Server	It was developed to provide security within the implementation while providing an encrypted path where common files used within the application are located or stored, usually outside the application's main directory.
2	API Gateway	It was developed to achieve authentication by identifying various resource requirements and declining requests that do not meet these requirements. Some of these requirements might be to authenticate and authorize the identity of the entity requesting access to the application. It also provides dynamic routing to other microservices hosting the requested services
3	Registry Service	It was developed to facilitate the registration of other microservices as they specify their host, port, node name, and other specific metadata.
4	IoT PoC	It was designed to provide the much-needed communication between layer 1 (edge platform) and layer 3 (blockchain) through the messaging system. It was also designed and developed to facilitate the functional requirements and interaction within the architecture.

Layer 2: Kafka

The following topics were created on Kafka and used for communication between the consumer and the producer. 'Userregistraton'—which is where the automatically registered Fabric organizations, users, and invoked Fabric CA details are published; 'Channel creation'— an established channel between the peers of the two organizations, with the payload also pushed on Kafka from Fabric, which is automatically invoked using the API created on Fabric; 'Channel join'—which aids the joining of the peers of the two organization created on Fabric. This ensures privacy between the two organizations, and also enforces that between the IoT devices; 'Transaction'—which deals with the exchange of information within the organizations and works in conjunction with the activities of the channel; 'Chaincode'—which facilitates the initiation of chaincode, although this was not actively used in this study; and 'Instantiate'—which instantiates the chaincode on the peers, and, with the help of the orderer, a consensus is reached by the peers of the two organizations on committing the transactions to the ledger.

A group was also created in Zookeeper to form the cluster. This is essential to create fault tolerance within our implementation as Zookeeper ensures topics are divided into several partitions, and manages the delegation of leaders and followers, and automatically assigns a new leader should the active partition fail for any reason. The Kafka producer bootstrap-server, Kafka admin properties bootstrap services, Kafka consumer bootstrap-

servers, and Kafka consumer group id on our Spring boot application and run on port 9092 are also configured

Layer 3: Hyperledger Fabric Component

Hyperledger Fabric 1.4.4 artifacts [56], i.e., the first long-time support version of Fabric at the time of this implementation, was used for the implementation. This research had already begun before the release of version 2.0. As earlier pointed out, in Section 3, several components form the foundation of the Fabric Network. NodeJS SDK of Fabric was used to interact with its modules. APIs were created, and a simple app was created on a dedicated port—4003 which also serves as the Hyperledger network adapter.

All APIs have an endpoint pointing to this port. It is important to state that Fabric is a containerized application, and Docker was used for its deployment. As such, using the YAML files, the network requirements were specified, with its artifacts to generate the network of two organizations. With this, a private network was created with each organization having its CA in the form of the Fabric CA, which the MSP of each organization uses to create identities.

Two organizations and two users per organization were specified (since we are using two virtual devices from EdgeX) alongside the admin for each of them. A bash script that starts the private network was created, and with the APIs designed, the private network starts automatically and generates each organization's artifacts. The Hyperledger Fabric network adapter facilitated this within the implementation. These artifacts include the users, enabled by Fabric CA and MSP, the peers, ledgers, and channel.

A NodeJS application was created where all the dependencies of our network were 'called' and referenced accordingly. For example, the kafka-node, and its dependencies, and all the REST endpoints. Node version 8.17.0 and npm version 6.14.5 are required for this Fabric network to run. Ultimately, the Fabric network created can serve as a testbed for any application to use while just making little modification where necessary, but the APIs created will work in any scenario. Figure 4 shows the interaction between Fabric, Kafka, and IoT POC microservices within this implementation.

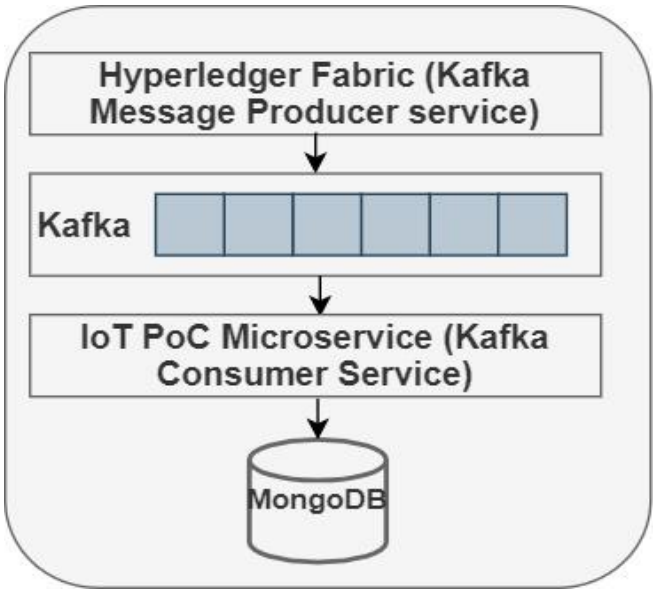


Figure 4. Interaction between Fabric, Kafka, and Microservice.

4.2. Implementation Approach

As earlier discussed in Section 3, by leveraging on the functionalities provided by EdgeX, such as device-virtual-service, two virtual devices were utilized: KMC.BAC-121036CE01, and JC-RR5.NAE9.ConfRoom.Padre.Island01. These devices were enabled through EdgeX with IoT PoC microservice facilitated through data transfer objects

(DTOs). For these devices to be registered, the EdgeX Docker containers and their UI components were started. Figure 5 shows our implementation approach between components of our architecture.

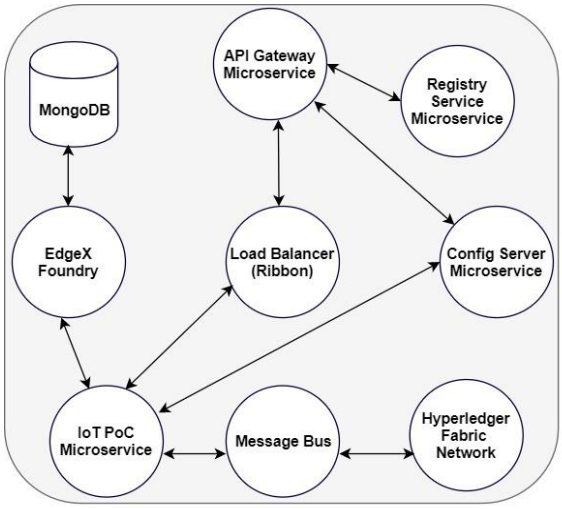


Figure 5. Implementation approach.

In Figure 5, a load balancer, Ribbon [57], which gives the application control over the collections of HTTP requests and TCP clients, was introduced. For simplicity, an overall flowchart of all steps taken within the application is shown in Figure 6.

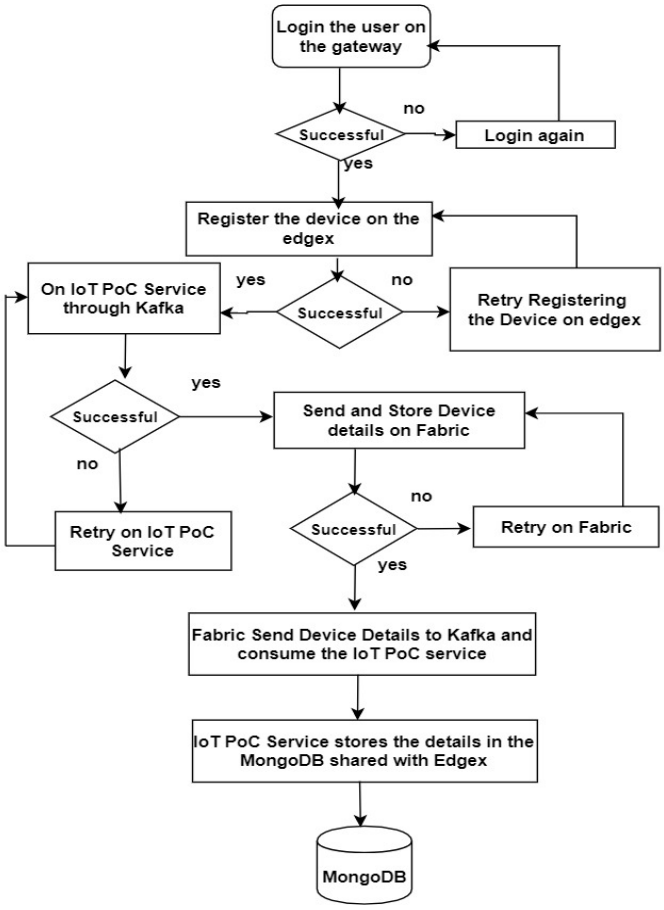


Figure 6. Flowchart showing steps taken for the implementation.

Figure 6 presents a detailed flowchart of what happened within the implementation approach. From Figures 5 and 6, the API gateway microservice facilitates the login process. At this stage, authentication and authorization are achieved by initiating the instance of the application through the Swagger implemented with it. A username and password are required to initiate this process, which provides authentication and authorization for the solution. After this is achieved, the devices are registered on EdgeX, but before this, EdgeX provides a UI that runs on port 4000 and requires registration by providing an email and password. This provided a second level of security.

Once devices are registered on EdgeX, through the topics on Kafka previously discussed, and with the Fabric endpoints (Hyperledger network adapter), the IoT PoC microservice made the device details from EdgeX available, and the APIs automatically created an identity for the devices on Fabric users within each organization. Since this is a test network developed for this implementation, it can be replaced at any time by changing it in the orderer.yaml file.

The IoT devices are uniquely identified on EdgeX as well as on Fabric. This is because EdgeX used ObjectID while Fabric, through its CA, and MSP gave identities in the form of certificates and secret keys. In addition, the devices are registered on users of each organization as facilitated by the Fabric network adapter. This provides security across our architecture and improves on the one provided by EdgeX and the one achieved through the microservices.

The devices which are uniquely identified in these organizations create interactions between their peers through the Fabric channel. This channel creates a data communication path between the devices, which enforces privacy, as only the devices on this channel can share data and be held accountable for this. Additionally, if a device shares data that resulted in any action being taken by another device (actuation), this activity can be traced back to the originating device. This accountability property enhanced by the immutability feature of Hyperledger Fabric will offer non-repudiation of actions between the devices, as the origin of data can be traced to each device.

Fabric internally used CouchDB for ledger storage (transactions and blocks), with instances stored across each peer of the organizations. This storage on the ledger of each peer provides consistent records across both organizations. As such, the content of each ledger of the peers is the same. With this, decentralized data storage is provided within the solution. For example, suppose a peer of an organization is attacked; in this case, the other peers will still have a valid ledger. As a result of this, the whole network will be safe from compromise because, before the content of a ledger on a peer can change, there has to be an initiation of transaction with a consensus reached by all the peers within the network. Furthermore, this decentralized data storage and data consistency provide data auditability because the data stored, which is immutable, can be queried.

4.3. Results

In the previous section, the implementation approach was presented alongside the details on how the connection was established across our architecture to address the challenges identified. Presented in this subsection are some results from our implementation. Figure 7 shows the output of the device registered on EdgeX. After devices are registered on EdgeX, a unique objectID is assigned. Fabric published the details of the device automatically registered on it to Kafka, and IoT PoC microservice consumes this and saves it on MongoDB, which updates the device details on EdgeX. This step keeps the details of the device's operations updated on MongoDB. Figure 8 shows the output of devices registration on both organizations of Fabric and presented in JSON format.

Figure 8 shows a sample of device registration on Fabric. JC.RR5.NAE9.ConfRoom.Padre.Island was registered on user btcii2 and btiic3 on Fabric organization 1 and 2, respectively, while device KMC.BAC121036CE was also registered on user btcii4, and

btcii5, respectively. In addition, Fauxton, a web UI for CouchDB usually used in development to visualize the content of the database on the ledger of the peers, was also enabled to access the secret key and certificate of users in each organization.

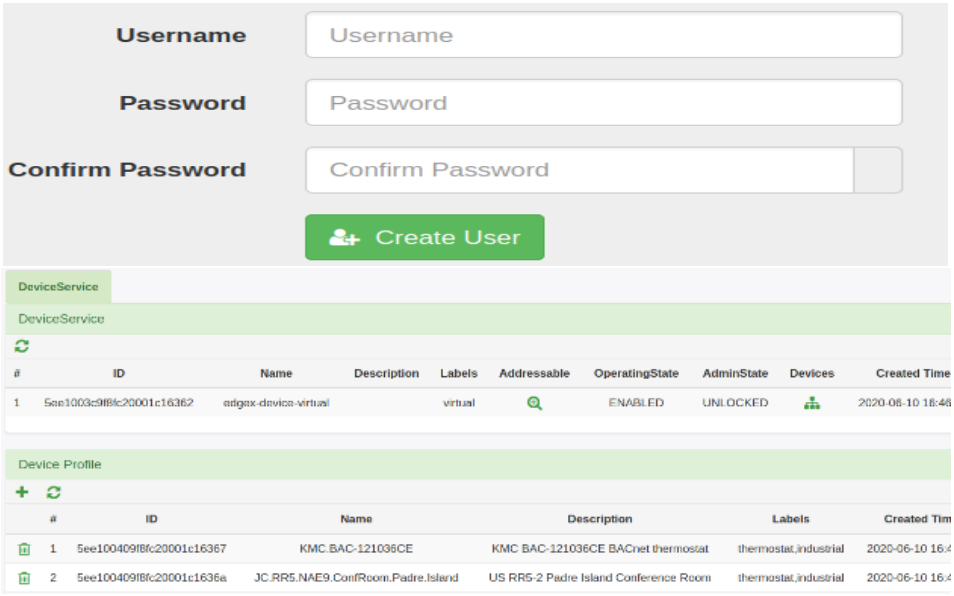


Figure 7. EdgeX usercreater UI and registered devices.


```
{
  "_id" : ObjectId("5ee122dda5a4cc3578831d5b"),
  "success" : true,
  "username" : "btcii2",
  "orgName" : "Org1",
  "deviceName" : "JC.RR5.NAE9.ConfRoom.Padre.Island01",
  "deviceCurrentValue" : -22.54,
  "id" : "JC.RR5.NAE9.ConfRoom.Padre.Island01registration1",
  "registerDate" : "2020-06-10T17:23:43.593Z",
  "message" : "btcii2 enrolled Successfully"
}
```


Figure 8. Devices registered on Fabric organizations.

Figure 9 shows a sample of the secret key and the certificates for admin and user of organization 1 and that of the device associated with it. This provided adequate security to the devices within the architecture since each device within the organization is uniquely identified. Therefore, communication is only established upon verification of the credentials of each user to which the devices are associated.

```
{
  "_id": "adminorg1",
  "_rev": "1-fe0271c11927a6c459372b275e32a94c",
  "member":
"{\"name\":\"adminorg1\",\"mspid\":\"Org1MSP\",\"roles\":null,\"affiliation\":\"\",\"enrollment
Secret\":\"\",\"enrollment\":{\"signingIdentity\":\"e7371d62f7c82a98be539f9ecc0ed72e0f2abddd706
f3e90c4adc2bb1cf3a33a\",\"identity\":{\"certificate\":\"-----BEGIN CERTIFICATE-----
\\nMIICBjCCAaygAwIBAgIUfRWgjAQSOA5spKZg3WkVzl/BD9UwCgYIKoZIzj0EAwIw\\nczEL
MAkGA1UEBhMCVVMxEzARBgNVBAgTCkNhbGlmb3JuaWExFjAUBgNVBAcTDVNh\\nbiBGcm
FuY2lzY28xGTAXBgNVBAoTEG9yZzEuZXhhbXBsZS5jb20xHDAaBgNVBAMT\\nE2NhLm9yZzEu
ZXhhbXBsZS5jb20wHhcNMjAwNjEwMTgwOTAwWhcNMjEwNjEwMTgx\\nNDAwWjAlMQ8wD
QYDVQQLEwZjbGllbnQxEjAQBgNVBAMTCWFkbWlub3JnMTBZMBMG\\nByqGSM49AgEGCCq
GSM49AwEHA0IABPKUaOcVwxROs18oWbrrEWgA+sDtXI7l2nnD\\nmhAwouUtRInFRwCtZcAFy
AEkvGe+CbsT6NqrOJhwBbpg8FNa57ujbDBqMA4GA1Ud\\nDwEB/wQEAwIHgDAMBgNVHRMB
Af8EAjAAMB0GA1UdDgQWBBTOnlJ8Z57PXnIAphKV\\ntOQR2VoChzArBgNVHSMEJDAigCAO
cpIk6LPzF4TIqTxbjvb0wckdnm5XfEXDMWNg\\nn+QAETAKBggqhkjOPQQDAgNIADBFAiEArp1
xWKGD3yK52Q85dUI1yu//V1s4r9lc\\n4T+3dCs5Iu4CIAZcGfLjQ73AdaUsoySoV8tdpoFhpgeMRQJy
Rc9nPrOw\\n-----END CERTIFICATE-----\\n\"}}}"
}
```

```
{
  "_id": "btcii2",
  "_rev": "1-a686eaba3a484f5b55e4c609e4f885ef",
  "member":
"{\"name\":\"btcii2\",\"mspid\":\"Org1MSP\",\"roles\":null,\"deviceName\":\"JC.RR5.NAE9.Conf
Room.Padre.Island01\",\"affiliation\":\"\",\"enrollmentSecret\":\"\",\"enrollment\":{\"signingIden
tity\":\"473c3c071cf9de1d8e85ea2cca415f02ee227bc96084d8fac7473eb39c1c91ec\",\"identity\":{\"certi
ficate\":\"-----BEGIN CERTIFICATE-----
\\nMIICkTCCAjegAwIBAgIUFK6hZ3gc2dF2C4uqo1i7kG8cTBswCgYIKoZIzj0EAwIw\\nczELMAk
GA1UEBhMCVVMxEzARBgNVBAgTCkNhbGlmb3JuaWExFjAUBgNVBAcTDVNh\\nbiBGcmFuY2
lzY28xGTAXBgNVBAoTEG9yZzEuZXhhbXBsZS5jb20xHDAaBgNVBAMT\\nE2NhLm9yZzEuZXhh
bXBsZS5jb20wHhcNMjAwNjEwMTgwOTAwWhcNMjEwNjEwMTgx\\nNDAwWjBDMTAwDQYD
VQQLEwZjbGllbnQwCwYDVQQLEwRvcmcxMBIGA1UECxMLZGVw\\nYXJ0bWVudDExDzANB
gNVBAMTBmJ0Y2lpMjBZMBMGByqGSM49AgEGCCqGSM49AwEH\\nA0IABDjk2HMHxS986wBz
VkBZvoq9l9RwUAgEH+GKkBX0sPUg90YkzQgxG7otgyNa\\nXYG4VLKv+ljk+ez5zi4A7ExSAOGjgd
gwgdUwDgYDVR0PAQH/BAQDAgeAMAwGA1Ud\\nEwEB/wQCMAAwHQYDVR0OBBYEFD5h
YprmDk8FBsi/R0CX6mjP8vUCMCsGA1UdIwQk\\nMCKAIA5ykiTos/MXhMipPFuO9vTByR2ebld8
RcMxY2Cf5AARMGkGCCoDBAUGBwgB\\nBF17ImF0dHJzIjp7ImhmLkFmZmlsaWF0aW9uIjoib3Jn
MS5kZXBhcnRtZW50MSIs\\nImhmLkVucm9sbG1lbnRJRCI6ImJ0Y2lpMiIsImhmL1R5cGUiOiJjbGllb
nQifX0w\\nCgYIKoZIzj0EAwIDSAAwRQIhAJfmB+F92ZNAJCZhtyfeNq9JDggamJeFh2vKmTZH\\
nVYxOAiAwmoVsZ6wDGZqXWCJ3snhVP8p0mSzq4D/5eSxLB9ss/w==\\n-----END CERTIFICATE--
---\\n\"}},\"sensorType\":\"Temperature\",\"deviceCurrentValue\":-22.54,\"actuatorName\":\"\"}"
}
```


Figure 9. Secret keys and certificates of admin and device on organization 1.

5. Performance Testing

It is important to conduct performance testing to check the viability of the implemented architecture. The application used microservices and Docker containers. First, the microservices designed can be replicated on the registry service (discovery service) to give our solution a fault tolerance feature. This also supports scalability. Kafka is also known to scale within an application with a fault tolerance feature. Fabric is also trusted to provide scalability within an application; this can be achieved by simply specifying the network details while setting up its network artifacts.

From the implementation, some metrics were gathered based on the underlisted:

(1) The latency of our solution based on service calls through API gateway using Zipkin. These metrics are necessary because all 'http' requests are routed through these microservices.
(2) A graphical representation of the resource utilization of some samples of Docker containers using cAdvisor. Considered here are few Docker containers that are used to process the core data generated within the architecture.
(3) The Hyperledger Fabric's transaction latency metrics which ultimately measures its performance according to the Linux Foundation white paper on Hyperlerger Fabric performance metrics [58].

Zipkin is used for monitoring Spring boot applications and for distributed tracing of 'http' requests while recording the timing or duration of when the request was granted. It also helps with data required for monitoring latency within a modular solution. These data are tied together into what is known as spans. It helps to trace both internal and external API calls for service. A unique trace ID is used to correlate this request. Zipkin runs in the background before the microservices are instantiated to measure the metrics adequately. It provides a user interface that assisted with the capturing of the 'http' request. Table 4 shows eight trace IDs, with two spans each, and their respective timing in milliseconds (ms).

It is important to note that there is no record of data to form a basis for comparison with our result, but we emphasize that the result presented is viable considering this application as a cross-domain application.

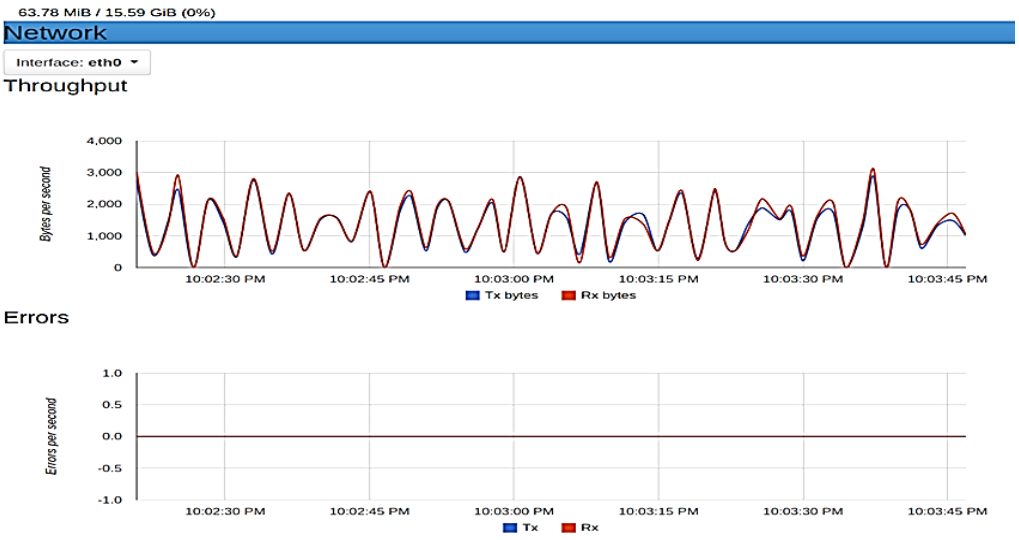
Examined next is the performance of the Docker containers used within the application. For instance, Hyperledger Fabric and EdgeX Foundry components are set up as Docker containers. This implementation was implemented using a system with the following configuration: Intel Core i7-3770 CPU @3.40 GHz, 16 GB RAM, and 64-bit Ubuntu 18.04.4 LTS operating system. Samples selected were a peer from one of the two organizations and the edgeX-core-metadata, where most of the transactions occur in EdgeX. The metrics for this performance evaluation is based on the undelisted:

(1) memory consumption;
(2) network throughput (Transfer—Tx, and Received—Rx (in bytes per second);
(3) transaction errors.

To achieve this, 'cAdvisor' (Container Advisor), a tool by Google that examines a container's resources usage and performance, was used. The samples of metrics taken using 'cAdvisor' are presented in histograms as seen in Figures 10 and 11.

Table 4. Resource call latency using Zipkin.

S/N	Trace ID	Total Spans	Duration (ms)
1	660782e3d5eb0967	2	368.167, 256.161
2	3d57f8d327ce6727	2	13.266, 7.340
3	6870d10dc9ae9a2d	2	9.235, 5.634
4	3ff7e2c1066649b3	2	8.998, 4.662
5	12b91717e27a71b1	2	8.967, 5.259
6	17cc84d3936a7437	2	8.686, 4.665
7	53c9e24c6e457101	2	7.517, 3.593
8	d4a04fa94c34ba7c	2	6.920, 3.792



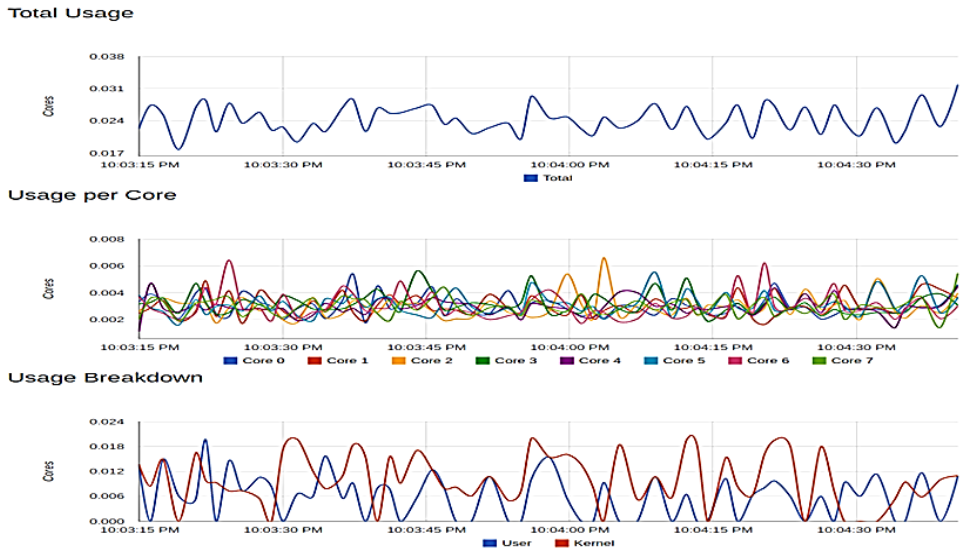


Figure 10. Resource usage for Peer0 and organization 2.

Figure 10 show the metrics for peer0 of organization 2 of Fabric. This container used 63.78 megabytes of memory (RAM). In terms of network throughput, 3000 bytes are transmitted and received per second at peak, with no error rate recorded in the transaction. The metrics obtained for peer1 of organization 1 at peak are as follows: 63.85 megabytes of memory consumed, 3200 bytes per second of data were transmitted and received with no data loss during transmission, which gives an error rate of zero.

The edge-core-metadata container from EdgeX (Figure 11) used as the edge computing platform was also examined. At peak, 20.72 megabytes of memory was consumed, 80,000 and 120,000 bytes of data were transmitted and received, respectively, with no error in transaction. For each of these containers, 0% of memory was consumed, and the CPU usage obtained was also very low despite an adequate amount of transmission in bytes per second taking place on the containers. No error was also recorded in the transmission of these data. These show that our application can effectively run on systems with minimal specifications.

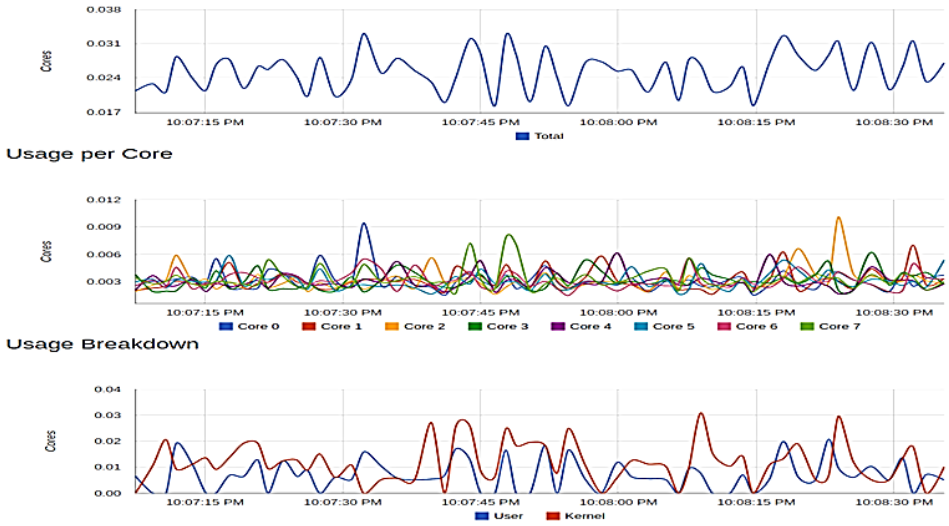


Figure 11. Resource usage of edgex-core-metadata.

Furthermore, to present Hyperlegder Fabric's performance based on transaction latency, we used the Hyperledger Caliper [59]. Hyperledger Caliper as a performance benchmark framework allows users to test different blockchain solutions for performance within a use case. At layer 3 of our architecture, the Fabric network adapter was developed to enhance the functionality of Hyperledger Fabric by automatically creating its network artifacts. For this implementation, two peers were created in each of the Fabric's organization that facilitates the automatic registration of devices registered on the Edge platform (Layer 1). We captured the transaction latency, a network-wide view of the amount of time taken for a transaction's effect to be usable across the network using Hyperlerger Caliper. This measurement includes when the device is registered on the edge to the point it is automatically associated with the users created on each organization and established data synchronization back to the edge, as facilitated by Layer 2 of the architecture.

Table 5 presents the summary of Fabric's organizations, peers, users, and associated devices used within our implementation, while Table 6 shows the longest commit time in seconds across each of the peers of both Fabric's organization for device registration and data communication read latency.

Table 5. Summary of fabric's artefact and associated devices.

Organizations	Peers	Users	Associated Devices
Organization 1	Peer 0, Peer 1	Admin, btcii2, btcii4	JC.RR5.NAE9.ConfRoom.Padre.Island, KMC.BAC121036CE
Organization 2	Peer 0, Peer 1	Admin, btcii3, btcii5	JC.RR5.NAE9.ConfRoom.Padre.Island, KMC.BAC121036CE

Table 6. Hyperledger Fabric Transaction Latency.

Organizations	Peers	Device Registration Commit Time (Seconds)	Data Communication Read Latency (Seconds)
Organization 1	Peer 0	7	18
	Peer 1	7	18
Organization 2	Peer 0	7	18
	Peer 1	7	18

In Table 6, the longest time taken for device registration and data communication/read latency, as initiated by the peers (0,1) of both organizations and committed to the ledger, are presented. It takes a total of 7 seconds for the devices to be issued with identities within the Fabric organization and committed to the Peer's ledger following the instantiation of the application. This comes after the provision of the IoT devices on the Edge platform, through the middleware (layer 2) and the Fabric network adapter.

In addition, it takes 18 seconds for data retrieval and communication back to the Edge platform from the ledgers through the messaging system of layer 2. This latency is essential to keep the Edge's database updated on the IoT devices' identities and aid synchronization and data consistency. A value of 7 and 18 seconds, respectively, for device registration and data communication read latencies for a multi-domain application indicate no blockchain overhead. This is due to the adequate steps and approaches taken to implement this novel architecture.

6. Conclusion and Future Work

Presented in this work is a three-layer architecture with unique features. A robust edge computing platform, called EdgeX and Hyperledger Ledger Fabric, were adapted and used within the architecture's design, development, and implementation. In addition, a Hyperledger Fabric network adapter was created that automatically produced the Fabric network and facilitated the connection established through microservices and Kafka.

IoT device data communication was established across this multi-domain application that addressed the challenges of edge computing in IoT platforms. This implementation addresses issues of central data storage by providing multiple device data storage on Fabric peers ledgers. Privacy was also achieved by creating a channel among Fabric organizations' peers that keeps the data shared among the devices private. Since the data in the ledgers are immutable, data auditability becomes possible with this application as it is easy to know which device initiates a transaction committed to the ledger. Therefore, our approach helps us to achieve non-repudiation of action among the devices because the origin of the data can be traced.

For security, the API gateway and Config Server microservices provide a first-level authentication and authorization, while second-level security is provided by EdgeX as users must be created before devices are registered. Finally, the third level of security was provided by Fabric CA and MSP, which assigned an enrolment, secret, public, and private key to admin of the organizations, while certificates and secret keys are also assigned to devices registered on Fabric users.

The registry microservice hosted on Eureka can replicate more microservices instances, thereby providing fault tolerance and scalability for our application. In addition, this work implemented a multi-domain platform as EdgeX Foundry and Hyperledger Fabric, which is used in architecture for IoT solutions for the first time. With this architecture implemented and tested in terms of its performance, we will further explore its potential in our subsequent studies. Areas that will be explored include how the capabilities of the IoT devices are maintained across the architecture, and the effectiveness of the security put in place within the implementation of the architecture will be further evaluated.

Author Contributions: Conceptualization, O.J.A.; methodology, O.J.A., J.R., J.S., M.G.-C. and Z.C.; software, O.J.A.; validation, J.R., J.S., M.G.-C. and Z.C.; formal analysis, O.J.A.; investigation, O.J.A., J.R., J.S., M.G.-C. and Z.C.; resources, O.J.A., J.R., J.S. and M.G.-C.; writing—original draft preparation, O.J.A.; writing—review and editing, J.R., J.S., M.G.-C. and Z.C.; visualization, O.J.A.; supervision, J.R., J.S., M.G.-C. and Z.C.; project administration, J.R., and Z.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by British Telecom and Invest Northern Ireland through BTIIC (British Telecom Ireland Innovation Centre) and the APC was funded by BTIIC.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data reported here were captured using the tools mentioned on the study. These data will vary based on the implementation. The details of these tools with required URL have been referenced within the paper.

Acknowledgments: This research is supported by the BTIIC (British Telecom Ireland Innovation Centre) project, funded by British Telecom and Invest Northern Ireland.

Conflicts of Interest: The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. IoT: Number of Connected Devices Worldwide 2012–2025. Statista. Available online: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/ (accessed on 10 July 2020).
2. Pal, D.; Funilkul, S.; Charoenkitkarn, N.; Kanthamanon, P. Internet-of-Things and Smart Homes for Elderly Healthcare: An End User Perspective. *IEEE Access* **2018**, *6*, 10483–10496, https://doi.org/10.1109/ACCESS.2018.2808472.
3. Laplante, N.L.; Laplante, P.A.; Voas, J.M. Stakeholder Identification and Use Case Representation for Internet-of-Things Applications in Healthcare. *IEEE Syst. J.* **2018**, *12*, 1589–1597, https://doi.org/10.1109/JSYST.2016.2558449.
4. Saleem, Y.; Crespi, N.; Rehmani, M.H.; Copeland, R. Internet of Things-Aided Smart Grid: Technologies, Architectures, Applications, Prototypes, and Future Research Directions. *IEEE Access* **2019**, *7*, 62962–63003, https://doi.org/10.1109/ACCESS.2019.2913984.
5. An, J.; Gall, F.L.; Kim, J.; Yun, J.; Hwang, J.; Bauer, M.; Zhao, M.; Song, J. Toward Global IoT-Enabled Smart Cities Interworking Using Adaptive Semantic Adapter. *IEEE Internet Things J.* **2019**, *6*, 5753–5765, https://doi.org/10.1109/JIOT.2019.2905275.
6. Dholu, M.; Ghodinde, K.A. Internet of Things (IoT) for Precision Agriculture Application. In Proceedings of the 2nd International Conference on Trends in Electronics and Informatics (ICOEI 2018), Tirunelveli, India, 11–12 May 2018; pp. 2018–2021.
7. Shafique, M.N.; Khurshid, M.M.; Rahman, H.; Khanna, A.; Gupta, D.; Rodrigues, J.J.P.C. The Role of Wearable Technologies in Supply Chain Collaboration: A Case of Pharmaceutical Industry. *IEEE Access* **2019**, *7*, 49014–49026, https://doi.org/10.1109/ACCESS.2019.2909400.
8. Al-Shargabi, B.; Sabri, O. Internet of Things: An Exploration Study of Opportunities and Challenges. In Proceedings of the 2017 International Conference on Engineering & MIS (ICEMIS), Monastir, Tunisia, 8–10 May 2017; pp. 1–4, https://doi.org/10.1109/ICEMIS.2017.8273047.
9. Tari, Z.; Yi, X.; Premarathne, U.S.; Bertok, P.; Khalil, I. Security and Privacy in Cloud Computing: Vision, Trends, and Challenges. *IEEE Cloud Comput.* **2015**, *2*, 30–38, https://doi.org/10.1109/MCC.2015.45.
10. El-Sayed, H.; Sankar, S.; Prasad, M.; Puthal, D.; Gupta, A.; Mohanty, M.; Lin, C.T. Edge of Things: The Big Picture on the Integration of Edge, IoT and the Cloud in a Distributed Computing Environment. *IEEE Access* **2017**, *6*, 1706–1717, https://doi.org/10.1109/ACCESS.2017.2780087.
11. Cao, K.; Liu, Y.; Meng, G.; Sun, Q. An Overview on Edge Computing Research. *IEEE Access* **2020**, *8*, 85714–85728, https://doi.org/10.1109/ACCESS.2020.2991734.
12. Elazhary, H. Internet of Things (IoT), Mobile Cloud, Cloudlet, Mobile IoT, IoT Cloud, Fog, Mobile Edge, and Edge Emerging Computing Paradigms: Disambiguation and Research Directions. *J. Netw. Comput. Appl.* **2019**, *128*, 105–140, https://doi.org/10.1016/j.jnca.2018.10.021.
13. Roman, R.; Zhou, J.; Lopez, J. On the Features and Challenges of Security & Privacy in Distributed Internet of Things. *Comput. Networks* **2013**, *57*, https://doi.org/http://doi.org/10.1016/j.comnet.2012.12.018.
14. Jiang, C.; Fan, T.; Gao, H.; Shi, W.; Liu, L.; Cérin, C.; Wan, J. Energy Aware Edge Computing: A Survey. *Comput. Commun.* **2020**, *151*, 556–580, https://doi.org/10.1016/j.comcom.2020.01.004.
15. Lan, L. Mechanism Based on Edge Computing for Internet of Things Real-Time Monitoring. *IEEE Access* **2019**, *7*.
16. Lan, L.; Shi, R.; Wang, B.; Zhang, L. An IoT Unified Access Platform for Heterogeneity Sensing Devices Based on Edge Computing. *IEEE Access* **2019**, *7*, 44199–44211, https://doi.org/10.1109/ACCESS.2019.2908684.
17. Khan, W.Z.; Ahmed, E.; Hakak, S.; Yaqoob, I.; Ahmed, A. Edge Computing: A Survey. *Futur. Gener. Comput. Syst.* **2019**, *97*, 219–235, https://doi.org/10.1016/j.future.2019.02.050.
18. Liu, F.; Tang, G.; Li, Y.; Cai, Z.; Zhang, X.; Zhou, T. A Survey on Edge Computing Systems and Tools. *Proc. IEEE* **2019**, *107*, 1537–1562, https://doi.org/10.1109/JPROC.2019.2920341.
19. Chowdhury, M.J.M.; Ferdous, M.S.; Biswas, K.; Chowdhury, N.; Kayes, A.S.M.; Alazab, M.; Watters, P. A Comparative Analysis of Distributed Ledger Technology Platforms. *IEEE Access* **2019**, *7*, 167930–167943, https://doi.org/10.1109/ACCESS.2019.2953729.
20. Böhme, R.; Christin, N.; Edelman, B.; Moore, T. Bitcoin: Economics, Technology, and Governance. *J. Econ. Perspect.* **2015**, *29*, 213–238.
21. Corda Open Source Blockchain Platform for Business. Available online: https://www.corda.net/ (accessed on 16 July 2020).
22. EOSIO—Blockchain Software Architecture. Available online: https://eos.io/ (accessed on 16 July 2020).
23. Home Ethereum.Org. Available online: https://ethereum.org/en/ (accessed on 16 July 2020).
24. MultiChain Open Source Blockchain Platform. Available online: https://www.multichain.com/ (accessed on 16 July 2020).

25. Home Quorum. Available online: https://www.goquorum.com/ (accessed on 16 July 2020).
26. Hyperledger Sawtooth—Hyperledger. Available online: https://www.hyperledger.org/use/sawtooth (accessed on 16 July 2020).
27. Hyperledger Fabric—Hyperledger. Available online: https://www.hyperledger.org/use/fabric (accessed on 16 July 2020).
28. Blockchain Showcase—Hyperledger. Available online: https://www.hyperledger.org/learn/blockchain-showcase (accessed on 10 July 2020).
29. Parikli, S.; Dave, D.; Patel, R.; Doshi, N. Security and Privacy Issues in Cloud, Fog and Edge Computing. *Procedia Comput. Sci. Sci.* **2019**, *160*, 734–739, https://doi.org/10.1016/j.procs.2019.11.018.
30. Kshetri, N. Can Blockchain Strengthen the Internet of Things? *Secur. IT* **2017**, *19*, 68–72, https://doi.org/10.1093/cercor/bhh040.
31. Siano, P.; De Marco, G.; Rolan, A.; Loia, V. A Survey and Evaluation of the Potentials of Distributed Ledger Technology for Peer-to-Peer Transactive Energy Exchanges in Local Energy Markets. *IEEE Syst. J.* **2019**, 1–13, https://doi.org/10.1109/JSYST.2019.2903172.
32. The Linux Foundation Project. Home—EdgeX Foundry. Available online: https://www.edgexfoundry.org/ (accessed on 10 July 2020).
33. Abdmeziem, R.M.; Tandjaoui, D.; Romdhani, I. Architecting the Internet of Things: State of the Art. *Robot. Sens. Cloud Springer* **2015**, *36*, 77–94, https://doi.org/10.1007/978-3-319-22168-7.
34. Martin Fernandez, C.; Diaz Rodriguez, M.; Rubio Munoz, B. An Edge Computing Architecture in the Internet of Things. In Proceedings of the 2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC), Singapore, 29–31 May 2018; pp. 99–102, https://doi.org/10.1109/ISORC.2018.00021.
35. Alanezi, K.; Mishra, S. An Edge-Based Architecture to Support the Execution of Ambience Intelligence Tasks Using the IoP Paradigm. *Futur. Gener. Comput. Syst.* **2021**, *114*, 349–357, https://doi.org/10.1016/j.future.2020.08.001.
36. Cicirelli, F.; Guerrieri, A.; Spezzano, G.; Vinci, A. A Cognitive Enabled, Edge-Computing Architecture for Future Generation IoT Environments. In Proceedings of the 2019 IEEE 5th World Forum on Internet of Things (WF-IoT), Limerick, Ireland, 15–18 April 2019; pp. 35–40, https://doi.org/10.1109/WF-IoT.2019.8767246.
37. Gheisari, M.; Pham, Q.V.; Alazab, M.; Zhang, X.; Fernandez-Campusano, C.; Srivastava, G. ECA: An Edge Computing Architecture for Privacy-Preserving in IoT-Based Smart City. *IEEE Access* **2019**, *7*, 155779–155786, https://doi.org/10.1109/ACCESS.2019.2937177.
38. Marjanovic, M.; Antonic, A.; Zarko, I.P. Edge Computing Architecture for Mobile Crowdsensing. *IEEE Access* **2018**, *6*, 10662–10674, https://doi.org/10.1109/ACCESS.2018.2799707.
39. Pace, P.; Aloi, G.; Gravina, R.; Caliciuri, G.; Fortino, G.; Liotta, A. An Edge-Based Architecture to Support Efficient Applications for Healthcare Industry 4.0. *IEEE Trans. Ind. Inform.* **2019**, *15*, 481–489, https://doi.org/10.1109/TII.2018.2843169.
40. Goyal, P.; Sahoo, A.K.; Sharma, T.K. Internet of Things: Architecture and Enabling Technologies. *Mater. Today Proc. Elsevier* **2020**, *34*, 719–735, https://doi.org/10.1016/j.matpr.2020.04.678.
41. Akkaoui, R.; Hei, X.; Cheng, W. EdgeMediChain: A Hybrid Edge Blockchain-Based Framework for Health Data Exchange. *IEEE Access* **2020**, *8*, 113467–113486, https://doi.org/10.1109/ACCESS.2020.3003575.
42. Bonnah, E.; Shiguang, J. DecChain: A Decentralized Security Approach in Edge Computing Based on Blockchain. *Futur. Gener. Comput. Syst.* **2020**, *113*, 363–379, https://doi.org/10.1016/j.future.2020.07.009.
43. Chuang, I.H.; Huang, S.H.; Chao, W.C.; Tsai, J.S.; Kuo, Y.H. TIDES: A Trust-Aware IoT Data Economic System with Blockchain-Enabled Multi-Access Edge Computing. *IEEE Access* **2020**, *8*, 85839–85855, https://doi.org/10.1109/ACCESS.2020.2991267.
44. Cui, L.; Yang, S.; Chen, Z.; Pan, Y.; Ming, Z.; Xu, M. A Decentralized and Trusted Edge Computing Platform for Internet of Things. *IEEE Internet Things J.* **2020**, *7*, 3910–3922, https://doi.org/10.1109/JIOT.2019.2951619.
45. Guo, S.; Dai, Y.; Guo, S.; Qiu, X.; Qi, F. Blockchain Meets Edge Computing: Stackelberg Game and Double Auction Based Task Offloading for Mobile Blockchain. *IEEE Trans. Veh. Technol.* **2020**, *69*, 5549–5561, https://doi.org/10.1109/TVT.2020.2982000.
46. Guo, S.; Hu, X.; Guo, S.; Qiu, X.; Qi, F. Blockchain Meets Edge Computing: A Distributed and Trusted Authentication System. *IEEE Trans. Ind. Inform.* **2020**, *16*, 1972–1983, https://doi.org/10.1109/TII.2019.2938001.
47. Li, M.; Yu, F.R.; Si, P.; Wu, W.; Zhang, Y. Resource Optimization for Delay-Tolerant Data in Blockchain-Enabled IoT with Edge Computing: A Deep Reinforcement Learning Approach. *IEEE Internet Things J.* **2020**, *7*, 9399–9412, https://doi.org/10.1109/jiot.2020.3007869.
48. Singh, S.K.; Rathore, S.; Park, J.H. BlockIoTIntelligence: A Blockchain-Enabled Intelligent IoT Architecture with Artificial Intelligence. *Futur. Gener. Comput. Syst.* **2020**, *110*, 721–743, https://doi.org/10.1016/j.future.2019.09.002.
49. Xu, J.; Wang, S.; Zhou, A.; Yang, F. Edgence : A Blockchain-Enabled Edge-Computing Platform for Intelligent IoT-Based DApps. *China Commun.* **2020**, *17*, 78–87.
50. Wang, R.; Tsai, W.T.; He, J.; Liu, C.; Li, Q.; Deng, E. A Video Surveillance System Based on Permissioned Blockchains and Edge Computing. In Proceedings of the 2019 IEEE International Conference on Big Data and Smart Computing (BigComp), Kyoto, Japan, 27 Feburary–2 March 2019; pp. 1–6, https://doi.org/10.1109/BIGCOMP.2019.8679354.
51. Apache Kafka. Available online: https://kafka.apache.org/ (accessed on 10 July 2020).
52. Kim, J.B.; Kwon, D.H.; Hong, Y.G.; Lim, H.K.; Kim, M.S.; Han, Y.H. Deep Q-Network Based Rotary Inverted Pendulum System and Its Monitoring on the EdgeX Platform. In Proceedings of the 2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), Okinawa, Japan, 11–13 Feburary 2019; pp. 34–39, https://doi.org/10.1109/ICAIIC.2019.8668979.

53. Sobecki, A.; Szymański, J.; Gil, D.; Mora, H. Deep Learning in the Fog. *Int. J. Distrib. Sens. Networks* **2019**, *15*, https://doi.org/10.1177/1550147719867072.
54. Xu, R.; Jin, W.; Kim, D. Microservice Security Agent Based on API Gateway in Edge Computing. *Sensors* **2019**, *19*, 1–17, https://doi.org/10.3390/s19224905.
55. Zhang, W.; Fan, H.; Zhang, Y.; Gao, Y.; Dong, W. Enabling Rapid Edge System Deployment with Tinyedge. In Proceedings of the SIGCOMM Posters and Demos '19: Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos, Beijing, China, 19–23 August 2019; pp. 104–106, https://doi.org/10.1145/3342280.3342323.
56. Introducing Hyperledger Fabric 1.4 LTS!—Hyperledger. Available online: https://www.hyperledger.org/blog/2019/01/10/introducing-hyperledger-fabric-1-4-lts (accessed on 10 July 2020).
57. Getting Started Client Side Load Balancing with Ribbon and Spring Cloud. Available online: https://spring.io/guides/gs/client-side-load-balancing/ (accessed on 4 November 2020).
58. The Linux Foundation Project. Hyperledger Blockchain Performance Metrics White Paper—Hyperledger. Available online: https://www.hyperledger.org/learn/publications/blockchain-performance-metrics (accessed on 24 September 2020).
59. The Linux Foundation. Fabric Hyperledger Caliper. Available online: https://hyperledger.github.io/caliper/v0.3.2/fabric-config/ accessed on 28 September 2020).